BigData UNIT1 QB

**Q1. BigData Characteristics.**

Big data is the collective name for the large amount of registered digital data and the equal growth thereof. The aim is to convert this stream of information into valuable information for the company. Yet it is not always clear what the definition of big data is and the term is used incorrectly.

To gain more insight into Big Data, IBM devised the system of the four Vs. These Vs stand for the four dimensions of Big Data: Volume, Velocity, Variety and Veracity.

**Volume:**

It is not surprising that Big Data is large in volume. It is estimated that we create 2.3 trillion gigabytes of data every day. And that will only increase. This increase is of course partly caused by the gigantic mobile telephone network. To give you an idea: six of the seven billion people in the world now have a mobile phone. Text and WhatsApp messages, photos, videos and many apps ensure that the amount of data increases significantly.

**Velocity**

Velocity, or speed, refers to the enormous speed with which data is generated and processed. Until a few years ago, it took a while to process the right data and to surface the right information. Today, data is available in real time. This is not only a consequence of the speed of the internet, but also of the presence of Big Data itself. Because the more data we create, the more methods are needed to monitor all this data, and the more data is monitored. This creates a vicious circle.

**Variety**

The high speed and considerable volume are related to the variety of forms of data. After all, smart IT solutions are available today for all sectors, from the medical world to construction and business. Consider, for example, the electronic patient records in healthcare, which contribute to many trillions of gigabytes of data. And that's not even talking about the videos we watch on YouTube; the posts we share on Facebook and the blog articles we write. When all parts of the world have the internet in the future, the volume and variety will only increase.

**Veracity**

How truthful Big Data is remains a difficult point. Data quickly becomes outdated and the information shared via the internet and social media does not necessarily have to be correct. Many managers and directors in the business community do not dare to make decisions based on Big Data.

**Q2. What are combiners? When should one use combiner in Map Reduce job?**

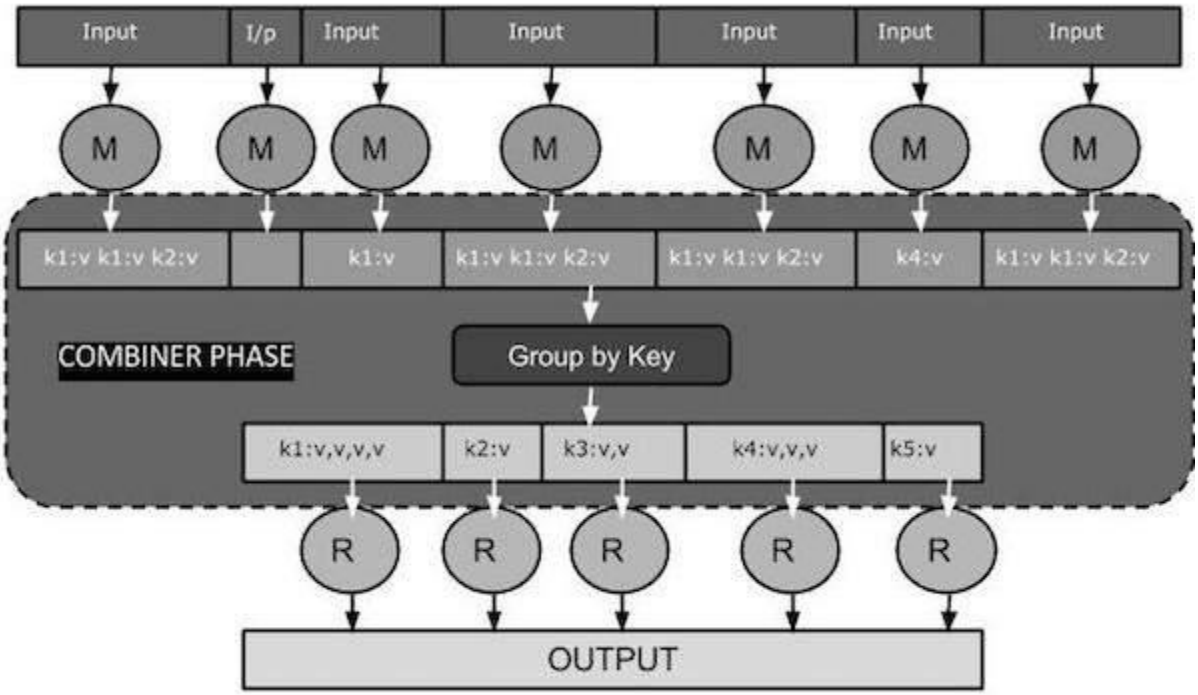
A Combiner, also known as a **semi-reducer,** is an optional class that operates by accepting the inputs from the Map class and thereafter passing the output key-value pairs to the Reducer class.

The main function of a Combiner is to summarize the map output records with the same key. The output (key-value collection) of the combiner will be sent over the network to the actual Reducer task as input.

Combiner

The Combiner class is used in between the Map class and the Reduce class to reduce the volume of data transfer between Map and Reduce. Usually, the output of the map task is large and the data transferred to the reduce task is high.

The following MapReduce task diagram shows the COMBINER PHASE.




How Combiner Works?

Here is a brief summary on how MapReduce Combiner works −

- A combiner does not have a predefined interface and it must implement the Reducer interface's reduce() method.
- A combiner operates on each map output key. It must have the same output key-value types as the Reducer class.
- A combiner can produce summary information from a large dataset because it replaces the original Map output.

Although, Combiner is optional yet it helps segregating data into multiple groups for Reduce phase, which makes it easier to process.

**Advantage of combiners:**

Reduces the time taken for transferring the data from Mapper to Reducer.

Reduces the size of the intermediate output generated by the Mapper.

Improves performance by minimizing Network congestion.

**Disadvantage of combiners:**

The intermediate key-value pairs generated by Mappers are stored on Local Disk and combiners will run later on to partially reduce the output which results in expensive Disk Input-Output.

The map-Reduce job can not depend on the function of the combiner because there is no such guarantee in its execution.

**Q3. Explain word count problem using map reduce**

MapReduce is a programming framework that allows us to perform distributed and parallel processing on large data sets in a distributed environment.

MapReduce consists of two distinct tasks — Map and Reduce.

As the name MapReduce suggests, reducer phase takes place after the mapper phase has been completed.

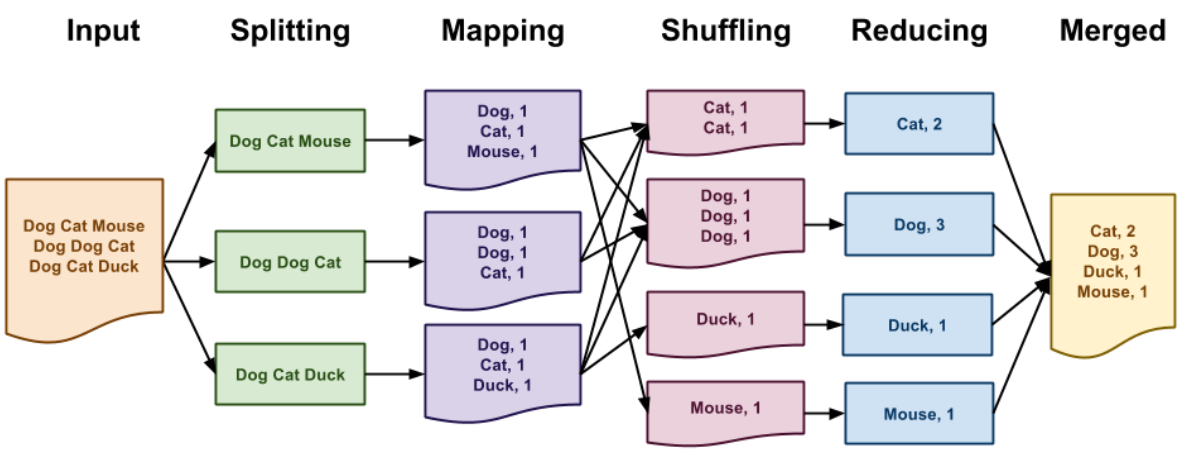
So, the first is the map job, where a block of data is read and processed to produce key-value pairs as intermediate outputs.

The output of a Mapper or map job (key-value pairs) is input to the Reducer.

The reducer receives the key-value pair from multiple map jobs.

Then, the reducer aggregates those intermediate data tuples (intermediate key-value pair) into a smaller set of tuples or key-value pairs which is the final output.

Example:




The input data is divided into multiple segments, then processed in parallel to reduce processing time. In this case, the input data will be divided into 3input splits so that work can be distributed over all the map nodes.

The Mapper counts the number of times each word occurs from input splits in the form of key-value pairs where the key is the word, and the value is the frequency.

For the first input split, it generates 3 key-value pairs: Dog,1; Cat,1; Mouse,1. For the second, it generates 3 key-value pairs:  Dog,1; Dog,1; Cat,1. For the second, it generates 3 key-value pairs: Dog,1; Cat,1; and Duck,1

It is followed by the shuffle phase, in which the values are grouped by keys in the form of key-value pairs. Here we get a total of 4 groups of key-value pairs.

The same reducer is used for all key-value pairs with the same key.

All the words present in the data are combined into a single output in the reducer phase. The output shows the frequency of each word.

Here in the example, we get the final output of key-value pairs as Cat,2; Dog,3; Duck,1; Mouse,1.

The record writer writes the output key-value pairs from the reducer into the output files, and the final output data is by default stored on HDFS.

**Q4. What is BigData? What is Hadoop? How are both linked?**

**What is big data?**

Big data is the collective name for the large amount of registered digital data and the equal growth thereof. The aim is to convert this stream of information into valuable information for the company. Yet it is not always clear what the definition of big data is and the term is used incorrectly.

**What is Hadoop?**

Apache Hadoop is an open source framework that is used to efficiently store and process large datasets ranging in size from gigabytes to petabytes of data. Instead of using one large computer to store and process the data, Hadoop allows clustering multiple computers to analyse massive datasets in parallel more quickly.

Hadoop consists of four main modules:

Hadoop Distributed File System (HDFS) – A distributed file system that runs on standard or low-end hardware. HDFS provides better data throughput than traditional file systems, in addition to high fault tolerance and native support of large datasets.

Yet Another Resource Negotiator (YARN) – Manages and monitors cluster nodes and resource usage. It schedules jobs and tasks.

Map Reduce – A framework that helps programs do the parallel computation on data. The map task takes input data and converts it into a dataset that can be computed in key value pairs. The output of the map task is consumed by reduce tasks to aggregate output and provide the desired result.

Hadoop Common – Provides common Java libraries that can be used across all modules.

**How are Hadoop and big data linked?**

In a fast-paced and hyper-connected world where more and more data is being created, Hadoop's breakthrough advantages mean that businesses and organizations can now find value in data that was considered useless.

Organizations are realizing that categorizing and analysing Big Data can help make major business predictions. Hadoop allows enterprises to store as much data, in whatever form, simply by adding more servers to a Hadoop cluster. Each new server adds more storage and processing power to the cluster. This makes data storage with Hadoop less expensive than earlier data storage methods.

With 90 percent of data being unstructured and growing rapidly, Hadoop is required to put the right Big Data workloads in the right systems and optimize data management structure in an organization. The cost-effectiveness, scalability and systematic architecture of Hadoop make it more necessary for organizations to process and manage Big Data.

**Q5. Explain hadoop architectural model/HDFS architecture with diagram.**

Hadoop File System was developed using distributed file system design. It is run on commodity hardware. Unlike other distributed systems, HDFS is highly faulttolerant and designed using low-cost hardware.

HDFS holds very large amount of data and provides easier access. To store such huge data, the files are stored across multiple machines. These files are stored in redundant fashion to rescue the system from possible data losses in case of failure. HDFS also makes applications available to parallel processing.

Features of HDFS

It is suitable for the distributed storage and processing.

Hadoop provides a command interface to interact with HDFS.

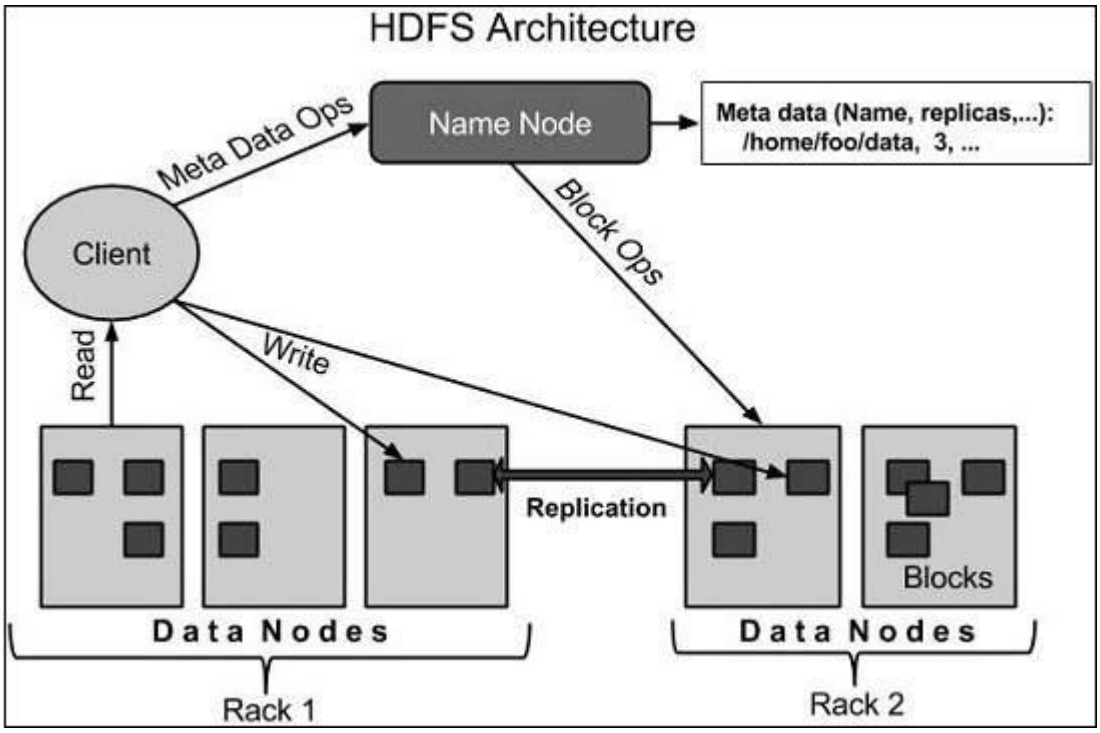
The built-in servers of namenode and datanode help users to easily check the status of cluster.

Streaming access to file system data.

HDFS provides file permissions and authentication.

**HDFS Architecture**

Given below is the architecture of a Hadoop File System.




HDFS follows the master-slave architecture and it has the following elements.

Namenode

The namenode is the commodity hardware that contains the GNU/Linux operating system and the namenode software. It is a software that can be run on commodity hardware. The system having the namenode acts as the master server and it does the following tasks:

Manages the file system namespace.

Regulates client's access to files.

It also executes file system operations such as renaming, closing, and opening files and directories.

Datanode

The datanode is a commodity hardware having the GNU/Linux operating system and datanode software. For every node (Commodity hardware/System) in a cluster, there will be a datanode. These nodes manage the data storage of their system.

Datanodes perform read-write operations on the file systems, as per client request.

They also perform operations such as block creation, deletion, and replication according to the instructions of the namenode.

Block

Generally the user data is stored in the files of HDFS. The file in a file system will be divided into one or more segments and/or stored in individual data nodes. These file segments are called as blocks. In other words, the minimum amount of data that HDFS can read or write is called a Block. The default block size is 64MB, but it can be increased as per the need to change in HDFS configuration.

Goals of HDFS

Fault detection and recovery : Since HDFS includes a large number of commodity hardware, failure of components is frequent. Therefore HDFS should have mechanisms for quick and automatic fault detection and recovery.

Huge datasets : HDFS should have hundreds of nodes per cluster to manage the applications having huge datasets.

Hardware at data : A requested task can be done efficiently, when the computation takes place near the data. Especially where huge datasets are involved, it reduces the network traffic and increases the throughput.

**Q6. List relational algebra operations. Explain any two using MapReduce.**

Selection (Select to from links where from='url1')

- Map: For each tuple t in R, test if it satisfies C. If so, produce the key-value pair (t, t). That is, both the key and value are t.
- Reduce: The Reduce function is the identity. It simply passes each key-value pair to the output.

Projection (Select to from links)

- Map: For each tuple t in R, construct a tuple t' by eliminating from t those components whose attributes are not in A. Output the key-value pair (t ' , t').
- Reduce: For each key t ' produced by any of the Map tasks, there will be one or more key-value pairs (t ' , t'). The Reduce function turns (t' , [t ' , t' , . . . , t' ]) into (t ' , t' ), so it produces exactly one pair (t ' , t') for this key t ' . (Duplicate Elimination)

Union

- Map: Turn each input tuple t either from relation R or S into a key-value pair (t, t). •
- Reduce: Associated with each key t there will be either one or two values. Produce output (t, t) in either case.

Intersection

- Map: Turn each input tuple t either from relation R or S into a key-value pair (t, t).
- Reduce: If key t has value list [t, t], then produce (t, t). Otherwise, produce nothing.

Difference (Minus) (Select to from links where from='url1' - select to from links where from='url2')

- Map: For a tuple t in R, produce key-value pair (t, name(R)), and for a tuple t in S, produce key-value pair (t, name(S)).
- Reduce: For each key t, do the following.
    1. If the associated value list is [name(R)], then produce (t, t).
    2. If the associated value list is anything else, which could only be [name(R), name(S)], [name(S), name(R)], or [name(S)], produce nothing.

**Example: Join By Map-Reduce**

- **Compute the natural join *R(A,B)* ⋈ *S(B,C)***
- *R* and *S* are each stored in files
- Tuples are pairs *(a,b)* or *(b,c)*

| A | B |
|---|---|
| $a_1$ | $b_1$ |
| $a_2$ | $b_1$ |
| $a_3$ | $b_2$ |
| $a_4$ | $b_3$ |

R

$\bowtie$

| B | C |
|---|---|
| $b_2$ | $c_1$ |
| $b_2$ | $c_2$ |
| $b_3$ | $c_3$ |

S

=

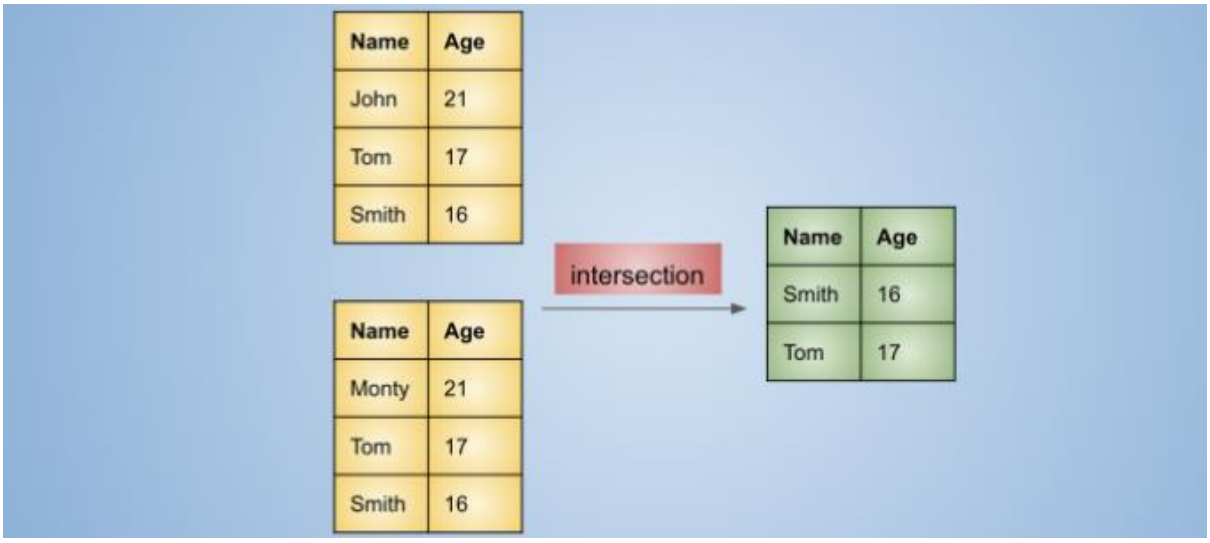
| A | C |
|---|---|
| $a_3$ | $c_1$ |
| $a_3$ | $c_2$ |
| $a_4$ | $c_3$ |

- **Use a hash function *h* from B-values to *1...k***
- **A Map process turns:**
  - Each input tuple *R(a,b)* into key-value pair *(b,(a,R))*
  - Each input tuple *S(b,c)* into *(b,(c,S))*
  - **Map processes** send each key-value pair with key *b* to Reduce process *h(b)*
    1. Hadoop does this automatically; just tell it what *k* is.
- Each **Reduce process** matches all the pairs *(b,(a,R))* with all *(b,(c,S))* and outputs *(a,b,c)*.

Grouping and Aggregtion

- Let assume that we group a relation R(A, B, C) by attributes A and aggregate values of B.
- Map: For each tuple (a, b, c) produce the key-value pair (a, b).
- Reduce: Each key a represents a group. Apply the aggregation operator θ to the list [b1, b2, . . . , bn] of B-values associated with key a. The output is the pair (a, x), where x is the result of applying θ to the list. For example, if θ is SUM, then x = b1 + b2 + . . . + bn, and if θ is MAX, then x is the largest of b1, b2,. . . , bn.

**Ex:-Interaction by map reduce**



**Q7. Discuss mapreduce function for natural join grouping, aggregation**

Natural Join by MapReduce

The idea behind implementing natural join via MapReduce can be seen if we look at the specific case of joining R(A, B) with S(B, C).

We must find tuples that agree on their B components, that is the second component from tuplesof R and the first component of tuples of S.

We shall use the B-value of tuples from either relation as the key.

The Map Function: For each tuple (a, b) of R, produce the key-value pair (b, (R, a)). For each tuple (b, c) of S, produce the key-value pair (b, (S, c)).

The Reduce Function: Each key value b will be associated with a list of pairs that are either of the form (R, a) or (S, c). Construct all pairs consisting of one

with first component R and the other with first component S, say (R, a) and (S, c). The output from this key and value list is a sequence of key-value pairs.

The key is irrelevant. Each value is one of the triples (a, b, c) such that (R, a) and (S, c) are on the input list of values for key b.

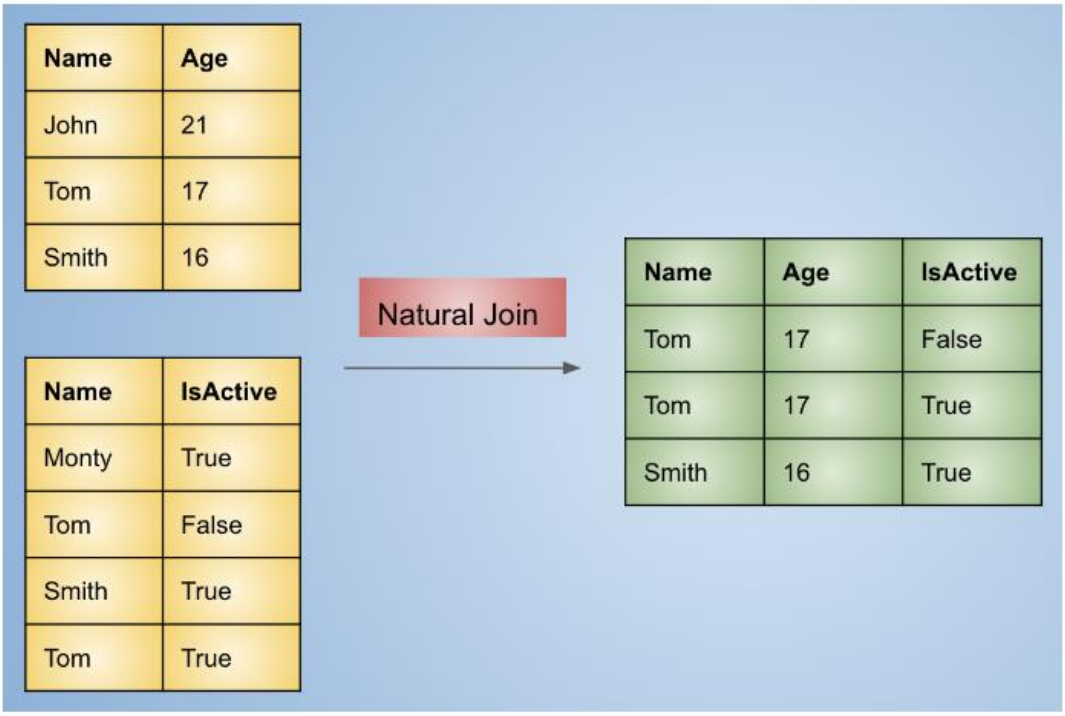
The same algorithm works if the relations have more than two attributes. You can think of A as representing all those attributes in the schema of R but

not S. B represents the attributes in both schemas, and C represents attributes only in the schema of S

The Reduce function looks at all the key-value pairs with a given key and combines those values from R with those values of S in all possible ways. From

each pairing, the tuple produced has the values from R, the key values, and the values from S.

- **Natural Join:** Merge two tables based on some common column. It represents the INNER JOIN in SQL. But the condition is implicit on the column that is common in both tables. The output will only contain rows for which the values in the common column matches. It will generate one row for every time the column value across two tables match as is the case below for `Tom`. If there were multiple `Tom` values in the table on the top in the image below, then four rows would have been created in the output table representing all the combinations.




**Grouping and Aggregation by MapReduce**

There is one grouping attribute (A), one aggregated attribute (B), and one attribute (C) that is neither grouped nor aggregated.

Let R(A, B, C) be a relation to which we apply the operator γA,θ(B)(R). Map will perform the grouping, while Reduce does the aggregation.

The Map Function: For each tuple (a, b, c) produce the key-value pair (a, b).

The Reduce Function: Each key a represents a group. Apply the aggregation operator θ to the list [b1, b2, . . . , bn] of B-values associated with key a. The

output is the pair (a, x), where x is the result of applying θ to the list.

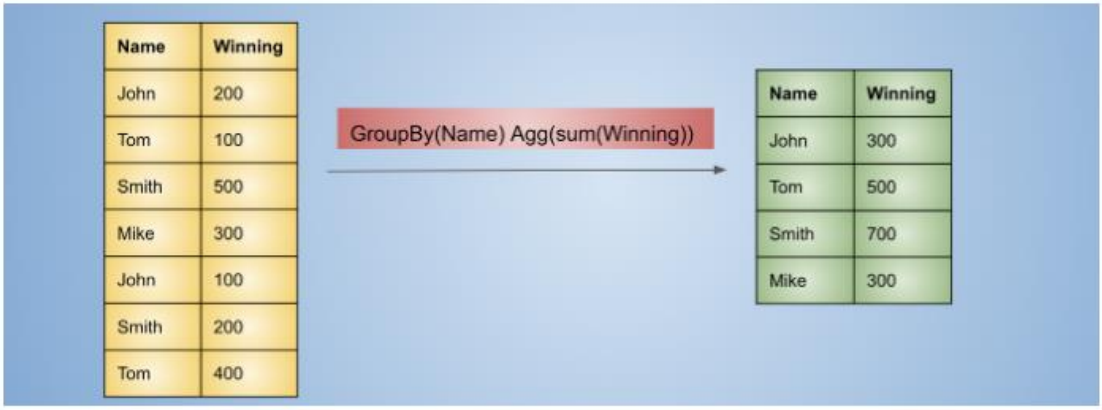
For example, if θ is SUM, then x = b1 + b2 + · · · + bn, and if θ is MAX, then x is the largest of b1, b2, . . . , bn.

If there are several grouping attributes, then the key is the list of the values of a tuple for all these attributes. If there is more than one aggregation, then

the Reduce function applies each of them to the list of values associated with a given key and produces a tuple consisting of the key, including components

for all grouping attributes if there is more than one, followed by the results of each of the aggregations.

- **Grouping and Aggregation:** Group rows based on some set of columns and apply some aggregation (sum, count, max, min, etc.) on some column of the small groups that are formed. This corresponds to GROUP BY in SQL.




Group By Name and take sum of the Winning

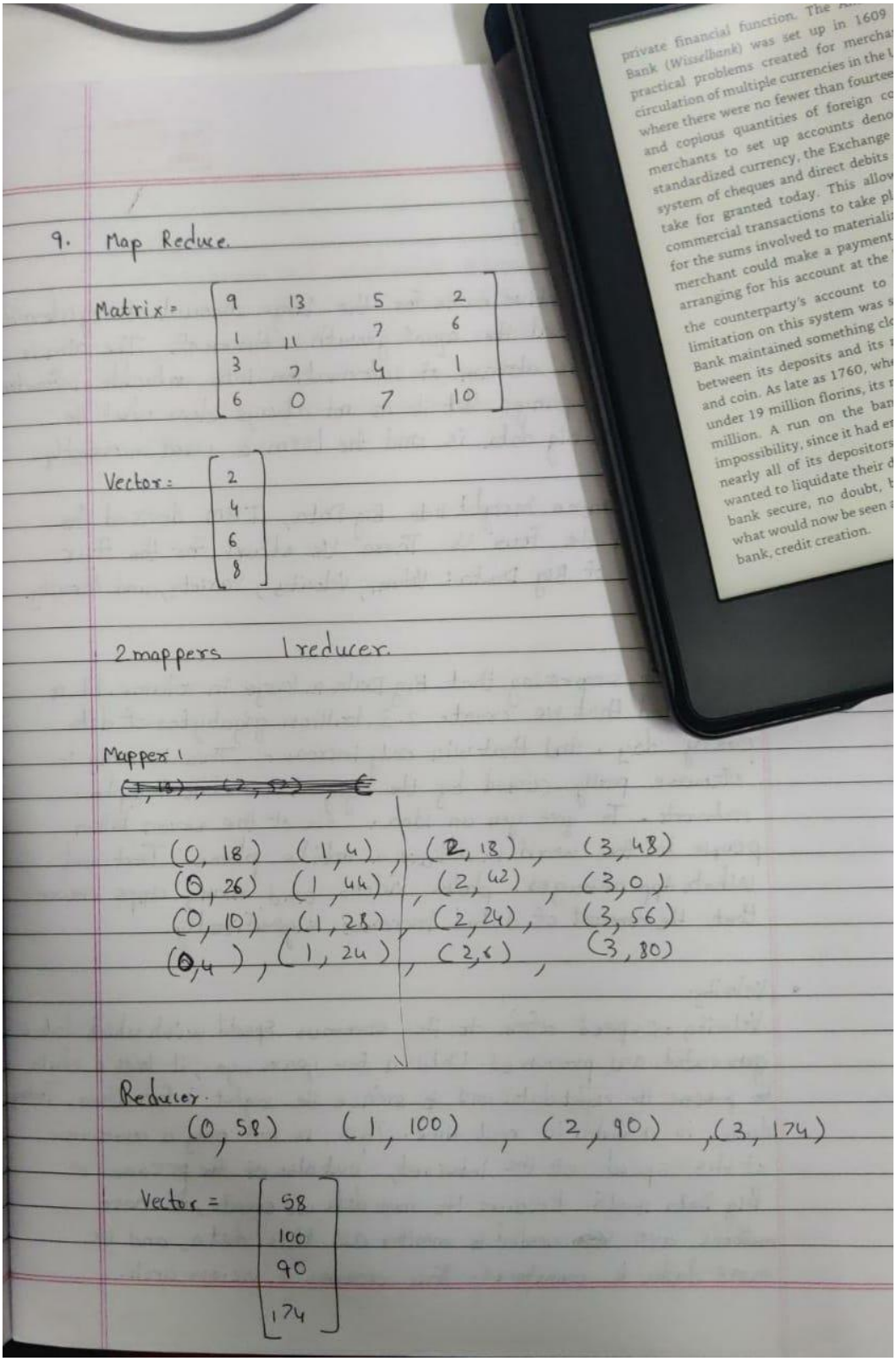
**Q8. Describe the operations of 'shuffle' and 'sort' in the Map reduce framework. Explain with the help of one example.**

Ans: The process of moving data from the mappers to reducers is shuffling. Shuffling is also the process by which the system performs the sort. Then it moves the map output to the reducer as input. This is the reason the shuffle phase is required for the reducers. Else, they would not have any input (or input from every mapper). Meanwhile, shuffling can begin even before the map phase has finished. Therefore this saves some time and completes the tasks in lesser time. MapReduce Framework automatically sorts the keys generated by the mapper. Therefore, before starting of reducer, all intermediate key-value pairs get sorted by key and not by value. It does not sort values transferred to each reducer. They can be in any order.

Sorting in a MapReduce job helps reducer to easily differentiate when a new reduce task should start. This saves time for the reducer. Reducer in MapReduce begins a new reduce task when the next key in the sorted input data is different from the earlier. Each reduce task takes key-value pairs as input and creates a key-value pair as output.

The crucial thing to note is that shuffling and sorting in Hadoop MapReduce is will not take place at all if you specify zero reducers (setNumReduceTasks(0)). If the reducer is zero, then the MapReduce job stops at the map phase. And the map phase does not comprise any kind of sorting (even the map phase is faster).

**Q9.**



9. Map Reduce.

Matrix =

| | | | |
|---|---|---|---|
| 9 | 13 | 5 | 2 |
| 1 | 11 | 7 | 6 |
| 3 | 7 | 4 | 1 |
| 6 | 0 | 7 | 10 |

Vector =

| |
|---|
| 2 |
| 4 |
| 6 |
| 8 |

2 mappers 1 reducer.

Mapper 1

(0, 18) (1, 4) (2, 18) (3, 48)
(0, 26) (1, 44) (2, 42) (3, 0)
(0, 10) (1, 28) (2, 24) (3, 56)
(0, 4) (1, 24) (2, 8) (3, 80)

Reducer.

(0, 58) (1, 100) (2, 90) (3, 174)

Vector =

| |
|---|
| 58 |
| 100 |
| 90 |
| 174 |

**10.Differentiate between scaling up and scale out. Which is better? What is achieved in NoSQL?**

Ans :

Schema-agnostic: NoSQL helps to store information without doing up-front schema design.

Nonrelational: The information in NoSQL is stored in the form aggregate. A single record stores every information about the transaction, including the delivery address.

Commodity hardware: NoSQL database is cheap and adding lower cost servers allows NoSQL databases to handle and scale more data.

Highly distributable: NoSQL uses the powerful, efficient architecture instead of the expensive monolithic architecture. The cluster of servers in a NoSQL database can be used to hold a single large database.

**Elastic Scaling**

The database administrators are always relying on a scale up. Purchasing the bigger servers as the database will increase load rather than scale out. The RDBMS will not scale out easily on commodity clusters. One of the major advantages of the new breed of NoSQL databases is that they are designed in such a way that they expand transparently to take advantage of new nodes, which significantly reduces the cost of commodity hardware.

**Sizable Data**

The transaction rates are growing tremendously in the current trend, hence the data that are being stored are also increased immensely. RDBMS capacity is increasing immensely to match this growth, but the data volume managed by a single RDBMS is becoming impossible for some companies. This drawback can be overcome by NoSQL as NoSQL systems help to handle large volume data such as Hadoop, outstrip which can be healed only by the biggest RDBMS.

**Flexible Data Models**

One of the major problems associated with RDBMS is Change management. Even the smallest change associated with the RDBMS data model should be carefully managed otherwise it reduces service levels. Whereas a NoSQL data model has fewer restrictions. Even the more rigidly defined BigTable-based NoSQL databases easily allow new column creation without creating much fuss.

**Cost Effective**

RDBMS rely on overpriced storage and proprietary server systems. While NoSQL databases commonly rely on clusters of cheap commodity servers to administer the tremendous data and transaction volumes.

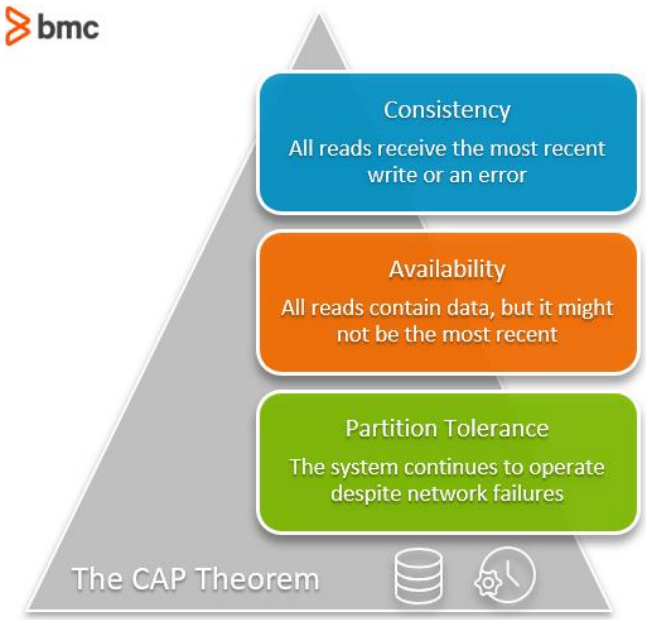
As the result, NoSQL is cost effective compared to RDBMS and allows to process and store more data at a much lower cost

| Scale out | Scale up |
|---|---|
| 1) Horizontal scaling, or scaling out or in, where you add more databases or divide your large database into smaller nodes, using a data partitioning approach called sharding, which can be managed faster and more easily across servers. | 2) Vertical scaling, or scaling up or down, where you increase or decrease computing power or databases as needed—either by changing performance levels or by using elastic database pools to automatically adjust to your workload demands. |
| 2) When new server racks are added in the existing system to meet the higher expectation, it is known as horizontal scaling. | 2) When new server racks are added in the existing system to meet the higher expectation, it is known as horizontal scaling. |
| 3) It expands the size of the existing system horizontally. | 3) It expands the size of the existing system vertically |
| 4) It is easier to upgrade | 4) It is harder to upgrade and may involve downtime |
| 5) .It is difficult to implement | 5) It is easy to implement |
| 6) It is costlier, as new server racks comprises of a lot of resources | 6) It is cheaper as we need to just add new resources |
| 7) It takes more time to be done | 7) It takes less time |

**Q11.**

**Q12. Explain cap theorem in detail.**

- The CAP theorem, originally introduced as the CAP principle, can be used to explain some of the competing requirements in a distributed system with replication. It is a tool used to make system designers aware of the trade-offs while designing networked shared-data systems.
- The three letters in CAP refer to three desirable properties of distributed systems with replicated data: consistency (among replicated copies), availability (of the system for read and write operations) and partition tolerance (in the face of the nodes in the system being partitioned by a network fault).
- The CAP theorem states that it is not possible to guarantee all three of the desirable properties – consistency, availability, and partition tolerance at the same time in a distributed system with data replication.




- The theorem states that networked shared-data systems can only strongly support two of the following three properties:
- **Consistency**
  - Consistency means that the nodes will have the same copies of a replicated data item visible for various transactions.
  - A guarantee that every node in a distributed cluster returns the same, most recent and a successful write.
  - Consistency refers to every client having the same view of the data. There are various types of consistency models. Consistency in CAP refers to sequential consistency, a very strong form of consistency.
- **Availability**
  - Availability means that each read or write request for a data item will either be processed successfully or will receive a message that the operation cannot be completed.
  - Every non-failing node returns a response for all the read and write requests in a reasonable amount of time. The key word here is “every”.
  - In simple terms, every node (on either side of a network partition) must be able to respond in a reasonable amount of time.

- **Partition Tolerance**
  - Partition tolerance means that the system can continue operating even if the network connecting the nodes has a fault that results in two or more partitions, where the nodes in each partition can only communicate among each other.
  - That means, the system continues to function and upholds its consistency guarantees in spite of network partitions. Network partitions are a fact of life.
  - Distributed systems guaranteeing partition tolerance can gracefully recover from partitions once the partition heals.

- The use of the word consistency in CAP and its use in ACID do not refer to the same identical concept.
- In CAP, the term consistency refers to the consistency of the values in different copies of the same data item in a replicated distributed system. In ACID, it refers to the fact that a transaction will not violate the integrity constraints specified on the database schema.

**Q13. Difference between Key Store & Graph Store.**

| Every data element in the database is stored as a key value pair consisting of an attribute name (or "key") and a value | A graph database focuses on the relationship between data elements. Each element is stored as a node |
|---|---|
| The speed depends upon the number of nested key value pairs. Generally O(1) | The speed depends upon the number of relationships among the database elements. |
| Low space consumption | High space consumption. Extra space required for storing relationships |
| Redis, Dynamo, Riak are some NoSQL examples of key-value store DataBases. | Neo4J, Infinite Graph, OrientDB, FlockDB are some popular graph-based databases. |

**Q14.Compare and contrast Master-Slave and Peer to peer distribution models**

- Peer-to-peer models may be more resilient to failure than master-slave models. Some master-slave distribution models have single points of failure that might impact your system availability, so you might need to take special care when configuring these systems.
- In the master-slave model, one node is in charge (master). When there's no single node with a special role in taking charge, you have a peer-to-peer distribution model. Figure shows you how each model works:

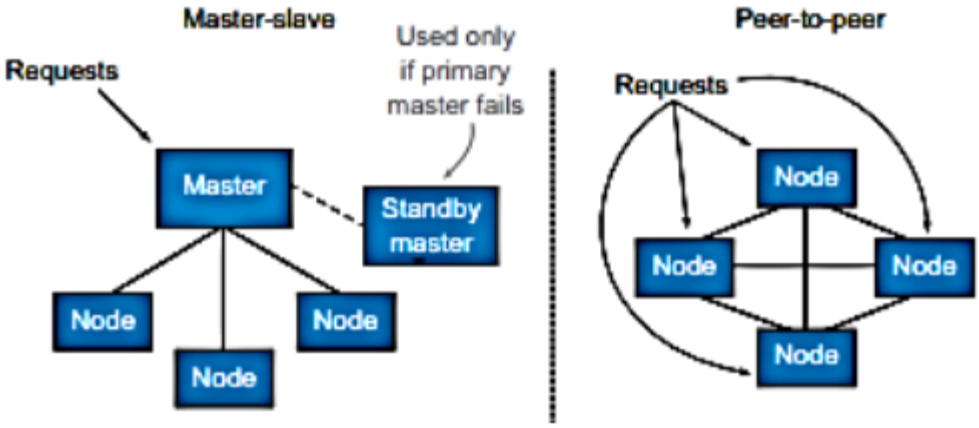



Figure: **Master-slave versus peer-to-peer**

- In the Figure, the panel on the left illustrates a master-slave configuration where all incoming database requests (reads or writes) are sent to a single master node and redistributed from there. The master node is called the NameNode in Hadoop. This node keeps a database of all the other nodes in the cluster and the rules for distributing requests to each node. The panel on the right

shows how the peer-to-peer model stores all the information about the cluster on each node in the cluster. If any node crashes, the other nodes can take over and processing can continue.

- With a master-slave distribution model, the role of managing the cluster is done on a single master node. This node can run on specialized hardware such as RAID drives to lower the probability that it crashes. The cluster can also be configured with a standby master that's continually updated from the master node. Peer-to-peer systems distribute the responsibility of the master to each node in the cluster. In this situation, testing is much easier since you can remove any node in the cluster and the other nodes will continue to function.
- Using the right distribution model will depend on your business requirements: if high availability is a concern, a peer-to-peer network might be the best solution. If you can manage your big data using batch jobs that run in off hours, then the simpler master-slave model might be best.

**Q15.Give one example and one real time application using column-family data store. Discuss its benefits.**